Turbulent Flow Simulations on HPC Systems
Final Project (Checkpointing, MPI IO, BFS)

Group 1: G. Chourdakis, J. Klicpera, M. Lin, T. Paula

January 21st 2016

1 Checkpointing with MPI I/O

1.1 Approach

In our implementation, the most fundamental data are gathered in a file at the end of the main loop, before creating the VTK files. Then, these data can be read at the beginning of a simulation, after initialization and before the main loop. The data that we store is the current timestep and time, as well as the pressure and velocity for every cell of each subdomain, excluding the ghost cells. Derived properties, such as the turbulent viscosity, are calculated again in the beginning of a restarted simulation, in order to minimize storage size. A plenty of new options are provided to the user, that we will discuss below.

1.2 Features

The following features have been implemented and tested:

- A checkpoint is exported every number of iterations chosen by the user, as well as after the end of the main loop.
- The path to the directory containing the checkpoints can be absolute or relative to the executable and does not need to preexist.
- The user can choose to automatically clean the checkpoint directory before writing new checkpoints.
- Only one binary file per timestep is used, written in parallel.
- A bash shell script to convert these binary files to human-readable form is provided.
- A tool to convert checkpoints to vtk files is also provided (created at compilation time), but mainly used for development purposes.
- The checkpoints are independent of the number of processes and can be used to restart a simulation with different partitioning.
- A simulation can be restarted from a specific or from the latest timestep available.
- The checkpoints can be used to continue a previous simulation or to start a completely new simulation, using the checkpoint only for initialization of the flowfield and setting the time to zero.
- Checkpointing has been tested for 2D and 3D, dns and turbulent simulations, for the Channel and BFS scenarios.

Some restrictions also exist:

- The checkpoint parameters must always be set. The directory and the prefix are mandatory, the rest are optional.
- The name of the checkpoints is always in the form `prefix.timestep`.
- The older checkpoints of the same simulation are not deleted. They can actually be useful in case of system failures or to restart a simulation with different parameters in case of divergence. Currently the user cannot change that.
- The `external32` representation is not fully supported in all MPI implementations yet, so performance was chosen over portability across systems. Please change the `native` to `internal` or (if supported) to `external32` in the code in case you need to transfer the files.
- The number of cells per dimension must be the same before and after a restart.

1.3 Options

The configuration has been extended as shown in the followig example snippet:

```
<checkpoint iterations="10" cleanDirectory="false">
    <directory>restart/</directory>
    <prefix>checkpoint</prefix>
</checkpoint>
<restart latest="true" startNew="false">restart/</restart>
```

The `checkpoint` node, together with the `directory` and `prefix` subnodes is mandatory. It sets the path and the name of the created checkpoint files. The `directory` must end with a path separator and it can be absolute or relative to the program executable. The `iterations` must be an integer and controls the frequency of the checkpoint creation. It is an optional flag that defaults to 1000. The `cleanDirectory` controls whether or not to delete all the files (not directories) under the `directory` path. *Be very careful NOT to delete any important files using this!*. This is also an optional flag, defaulting to `false`.

The `restart` node is optional and, if present, defines a file from which the simulation should restart from. Both the included flags are optional and default to `false`. In case the `latest` flag is set to `true`, the program looks for the checkpoint with the largest timestep number in the name. In case the `startNew` flag is set to `true`, the checkpoint is read to initialize the flowfield, but the time is set to zero.

Apart from these, the `vtk` node now also accepts an optional flag `active="true"` to enable/disable the VTK output. This defaults to `false`.

```
<vtk interval="1.0" active="true">output/bfs</vtk>
```

1.4 File format

We use only one binary file, in which all the processes write concurrently, using the I/O functionality of MPI. This contains a header with the timestep (`sizeof(int)`) and the time (`sizeof(FLOAT)`). In our setup, the header is 12 bytes long. The rest of the file contains the cell data for each point, in the format p, u, v, w (all of the type `FLOAT`). The points are traversed in an x-y-z order (z being the fastest index). The file is currently written using the `native` representation.

In order to read the restart file, the `hexdump` tool can be used in Linux. A script to facilitate reading our special format is provided. It can be called like this:

```
> ./bin2ascii checkpoint num_of_cells_to_convert [dimensions]
```

1.5 Implementation aspects

The checkpointing functionality is implemented mainly into the `Checkpoint.cpp` file and the respective header. A class `Checkpoint` is defined, that contains the functions:

```
/** Reads a checkpoint file
* @param timeStep the restart timestep
* @param time the restart time
*/
void read ( int& timeStep, FLOAT& time );

/** Creates a checkpoint file
 * @param timeStep the current timestep
 * @param time the current time
 */
void create ( int timeStep, FLOAT time );

/** Cleans the restart directory
*/
void cleandir ();
```

Since there are some differences between the 2D and 3D cases, the source file is quite long. The constructor of the function sets the parameters used for the reading and writing of files. The main view of the files is structured using a `subarray` MPI file type that needs to be committed before it is used. The subarray is a 3D array for the 3D case, but with 4 times longer z-dimension, for storing the p, u, v, w. The header is written using the `MPI_File_write()` and read using the `MPI_File_read_at()`, only by the master process. Next, a view of the file is set after the header. The data are aggregated to a `localarray` before being written, but they are directly assigned at reading. After everything is written/read, the file is closed. These functions are integrated into the Simulation class.

1.6 Performance measurements

The writing time for different simulation setups was measured to get an overview over the performance advantages by MPI IO. Table 1 shows the writing time and data size for different data representations using a simulation with a $80 \times 40 \times 40$ grid and $8 \times 1 \times 1$ processes. The other simulation parameters are irrelevant for the writing performance. 10 simulations were run, creating 25 checkpoints each.

The table 1 shows that all data representations create a file of the exact same size. Compared to the native representation, the runtime for writing in the IEEE standardized external32 format is about equally fast and the internal representation provided by the MPI implementation (i.e. Intel MPI) is significantly slower. Since the external32 format has some issues in some implementations (namely OpenMPI), we use the native format.

Table 1: Performance data for different file data representations.

Representation	Writing time [ms]	data size [MiB]
Native	45 ± 20	4.0
Internal	52 ± 29	4.0
External32	42 ± 24	4.0

After evaluating the data representation, the performance of the checkpoint writing method was also measured for different domain sizes and numbers of processes. A simple timing method using the system time was used for these measurements. Due to the short time the method needs for execution, this measurement is greatly influenced by

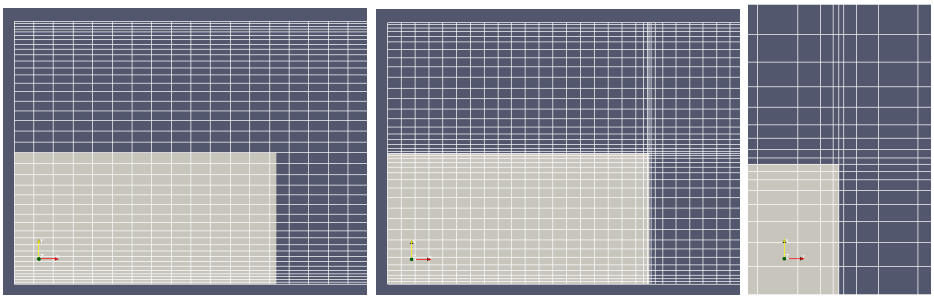
load imbalancing and the associated synchronization and waiting time needed by MPI. Therefore, the scaling tests produced mostly random results. This is why these results are omitted in this report. This problem could be circumvented by using a more sophisticated performance measurement tool like Scalasca.

This issue also influenced the performance measurement for different file data representations. For this test a larger sample was used and the influence was mostly averaged out, though.

2 Mesh Optimization

The `stretched` mesh type previously used for channel flow is not suitable for the backward facing step scenario as can be seen in Figure 1a. First, the step geometry is not perfectly captured by the mesh in most cases. Second, there is no refinement of the mesh towards the two walls of the step and thus the boundary layers at these walls are most certainly not resolved correctly. To overcome both issues a new mesh type called `bfs` was implemented and used for the final simulations.

The `bfs` mesh type that can be seen in Figure 1b treats the domain as two channels stacked in y-direction. In both channels the cells are distributed in y-direction with respect to the familiar tanh-stretching law. The number of cells for each channel is chosen in such a way that the size of the wall-nearest cells is as similar as possible in both channels. In x-direction a nearly uniform cell distribution is used. However, the edge of the step is always captured by a cell boundary. Furthermore, the cells directly left and right of this edge are split into three smaller cells that are distributed with respect to the tanh-stretching law (see Figure 1c). The z-distribution of the cells is the same as for the `stretched` mesh type since there are no additional walls parallel to the z-axis.



(a) `stretched` mesh type (b) `bfs` mesh type (c) `bfs` detail

Figure 1: Comparison of the `stretched` and `bfs` mesh types around the step (gray area)

Moreover, both the `stretched` and `bfs` mesh types are enhanced by accepting different stretching parameters δ_S for every direction. This allows to better adjust the mesh to a specific scenario.

Figure 2 shows the velocity profiles above the trailing edge of the step for the meshes from Figure 1. It can be seen that the `stretched` mesh does not allow the boundary layer to develop on top of the step, whereas the new mesh type does.

3 Final Case

Backward facing step (BFS) is a common benchmark scenario for validating CFD code. In this report, the simulation results of BFS is compared with the provided experimental data. The geometry of this specific BFS scenario is shown in Figure 3.

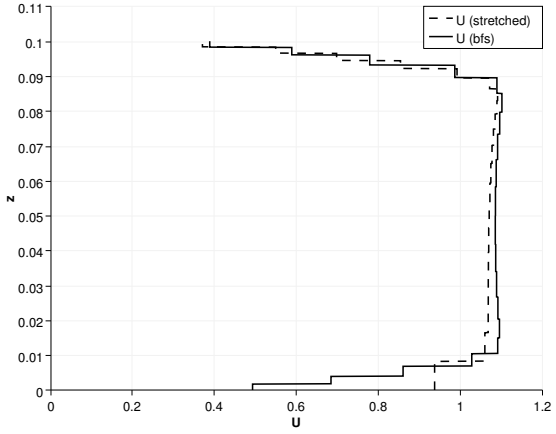


Figure 2: Velocity profiles above the trailing edge of the step on different meshes

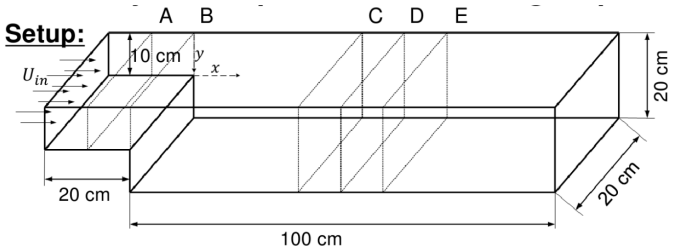


Figure 3: Geometry of the BFS scenario

3.1 Solver Settings

We use the Flexible Generalized Minimal Residual Method (fGMRES) solver provided in PETSc to solve this problem. In order to improve the convergence rate, we use the Incomplete LU (ILU) preconditioner for the solver. As we can see in Figure 4, faster convergence can be achieved by using more factorization levels for the ILU preconditioning, although it also takes more computation time per iteration at the same time. Therefore by balancing these 2 factors we choose the factorization level of 4 in the preconditioning as the optimal choice.

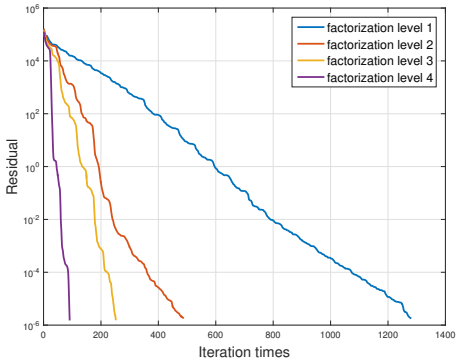


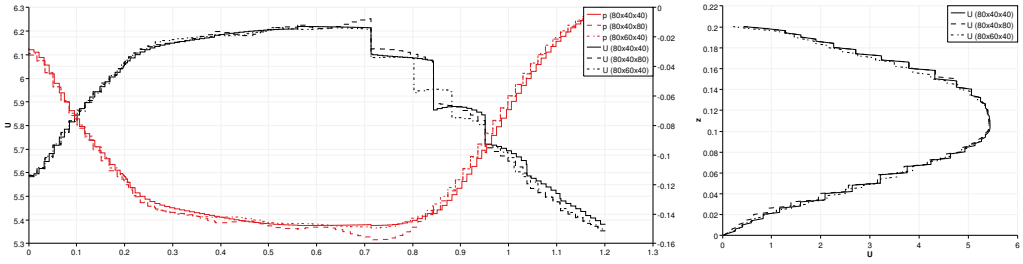
Figure 4: Convergence rate of PETSc solver with different preconditioning parameter

3.2 Mesh Setup

The domain is discretized by $80 \times 40 \times 40$ cells. The cells are distributed with respect to the `bfs` mesh type (see Section 2) using 1.7, 1.5 and 1.5 as stretching parameter in x-, y- and z-direction, respectively.

The accuracy of the simulation results is mostly influenced by the discretization in y-

and z-direction and the resulting resolution at the channel/step walls. A larger number of cells, however, leads to smaller cells and thus a smaller timestep. In order to find a trade-off we also tested other discretizations. Figure 5 shows velocity and pressure profiles of the $Re = 50160$ case along the line connecting the center points of the inlet and outlet faces of the channel as well as the velocity profile at the channel outlet. The profiles are shown for the mesh described above as well as for two refined meshes. The first variant uses 80 cells in z-direction while the second uses 60 cells in y-direction. The stretching parameters are unchanged. Besides different jumps in the profiles due to different positions of the cell boundaries, the profiles for all three meshes coincide. Since there was no gain from a finer mesh except smoother profiles, the $80 \times 40 \times 40$ variant was chosen for the final test. Furthermore, we tested higher stretching parameters δ_S, especially the previously hardcoded δ_S. However, stronger stretching does not only produce a finer resolution at the wall but also a coarser mesh away from the wall. Since the backward facing step scenario investigated here features flow structures apart from boundary layers, a moderate stretching is more suitable.



(a) Velocity and pressure profiles along the line connecting the center points of inlet and outlet face

(b) Velocity profiles shortly before the outlet

Figure 5: Velocity and pressure profiles for different cell numbers ($Re = 50160$ case)

Nevertheless, the mesh resolution at the wall has to be fine enough in order to correctly represent the boundary layer since it is not modeled by wall functions. The y^+ values of the wall-nearest cells give a good measure to assure this. Figure 6 shows these values for the mesh we used. Only at the inlet, where no boundary layer has developed yet, unfavorable values between 2 and 3. In the rest of the domain the values are close to 1 or smaller and thus suitable for resolving the boundary layer.

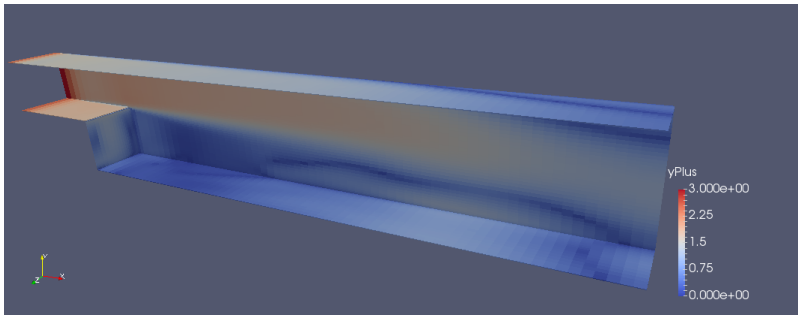


Figure 6: y^+ values of the wall-nearest cells ($Re = 50160$ case)

3.3 Estimating the reattachment length

Figure 7 shows an example of the streamline of the simulation case $Re = 50160$. We can see that a recirculation bubble is formed in the downstream region after the backward facing step until the main flow reattaches to the wall.

The flow is separated from the wall if the wall shear stress vector points upstream. In order to investigate the location of the reattachment point, we therefore plot the X

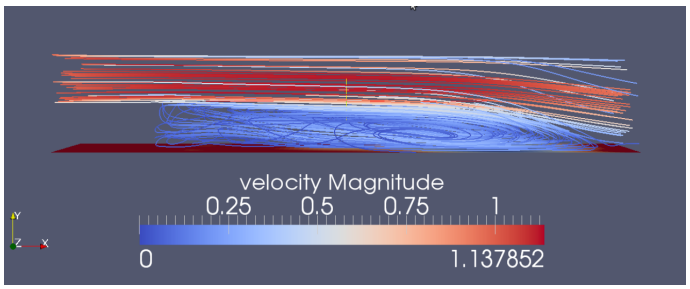
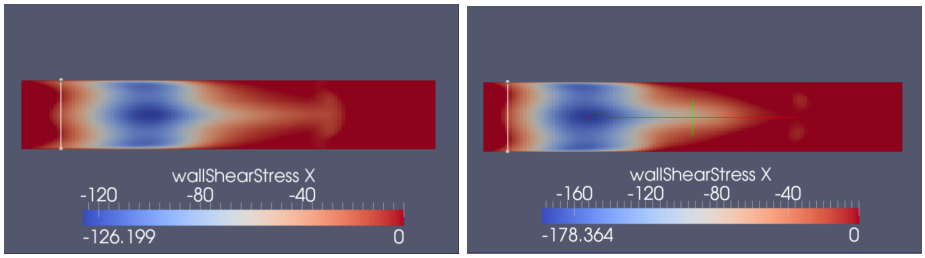
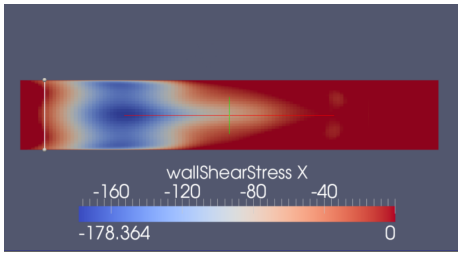


Figure 7: Streamline of the BFS scenario

component of the wall shear stress τ_w at the bottom of the channel. The reattachment point is estimated as the location where τ_w becomes positive after it was negative and thus $\tau_w = 0$ (Figure 8, τ_w is clipped at 0).



(a) $Re = 30317$: reattach at $88cm$ (b) $Re = 47606$: reattach at $93cm$



(c) $Re = 50160$: reattach at $94cm$

Figure 8: Wall shear stress (X component) at the bottom

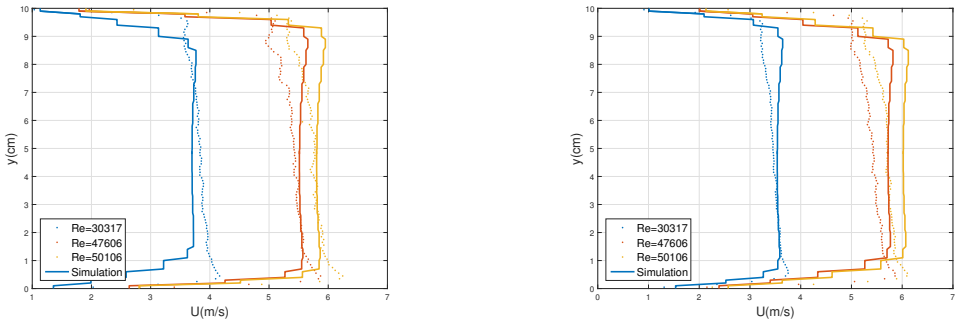
3.4 Comparison with the experimental data

In this section we will compare the simulation results with the experimental data at 5 different locations ($x = -10\,\text{cm}, x = 0\,\text{cm}, x = 40\,\text{cm}, x = 50\,\text{cm}, x = 60\,\text{cm}$).

As we can see from Figure 11, at the first two locations ($x = -100mm, x = 0mm$), the flow profile of the simulation result matches the overall shape of the experiment. But the experimental data is more sensitive to the asymmetry of the flow field further downstream while the simulation result fails to capture it.

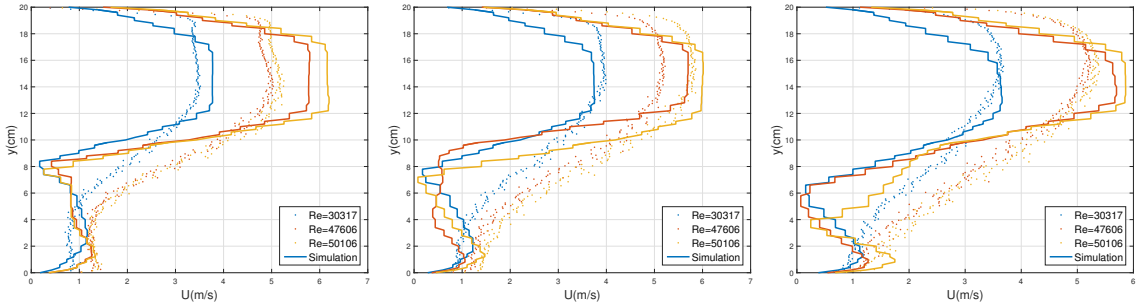
After the backfacing step, while in the upper part of the flow field the simulation results for all Reynolds numbers exhibit similar behavior to the experimental data, in the low part of the field the reults start to deviate from the experimental data. At the recirculation region (roughly between 2cm to 8cm to the wall), the simulation result shows a much more significant reverse flow than the experiment data. It suggest that the current simulation overestimate the size of the bubble and the reattachment length.

Such deviation is mainly due to the incapability of the simple mixing length model to handle the turbulence of the shear flow after the backward facing step. Figure 11a shows the streamline release from the edge of the BFS colored by the turbulent viscousity. After the flow leaves the BFS, according to the current turbulence model, the mixing length, as a rough representation of the turbulent scale, is still estimated by the turbulent flat



(a) $x = -10cm$ (b) $x = 0cm$

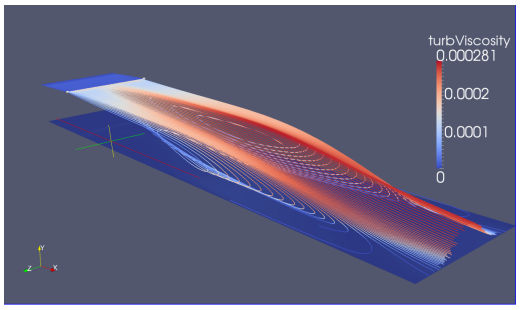
Figure 9: Velocity profile at the intersection planes before BFS



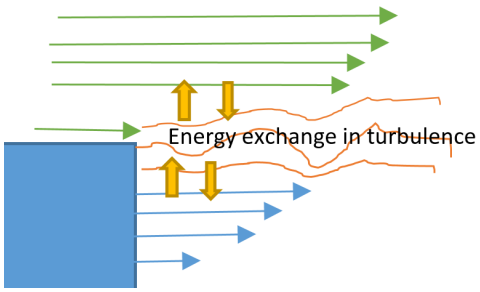
(a) $x = 40cm$ (b) $x = 50cm$ (c) $x = 60cm$

Figure 10: Velocity profile at the intersection planes after BFS

plate boundary layer thickness, which fails to include the extra turbulence produced by the shearing between two layers of the flow with different velocity, thus underestimate the turbulent viscosity here. Such underestimation hampers the energy transfer by turbulence between the upper and the lower part of the flow (Figure 11b), which happens in the actual flow, and as a result, delays the reattachment of the main flow in the simulation.



(a) Streamline from BFS edge (colored by ν_{turb})



(b) Kinetic energy transfer after BFS

Figure 11: Turbulence after BFS

We can expect that the result can be further improved by using more advanced turbulence models and simulation methods, such as large eddy simulation.